Introduction to control systems
TI0118

PROJECT WORK

STUDENT NAME: PEDRO LEINOS FALCAO CUNHA
STUDENT NUMBER: 542114
STUDENT NAME: KELVIN LEANDRO MARTINS
STUDENT NUMBER: 540006

TASK 1 - SYSTEM MODELLING

Consider the two tanks system arranged as shown in Figure 1.

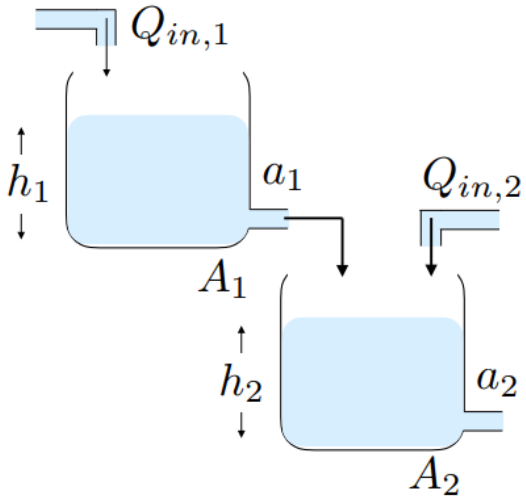

Figure 1: Two-tank system for task 1.

The volumetric flow rates $Q_{in,1}$ and $Q_{in,2}$ represent the inputs to the system, while the measured variable (output) is the fluid level h_2 in the second tank. The areas of the two tanks are A_1 and A_2, while a_1 and a_2 are the areas of the orifices indicated in Figure 1. The flow rate $Q_{in,2}$ is a second input to the system that can represent a possible disturbance to the second tank. The fluid is perfect (no shear stresses, no viscosity, no heat conduction), and subject only to gravity. The tanks are full of water (incompressible fluid) and the external pressure is constant (atmospheric pressure).

1. Obtain a state-space model of the system, where $x_1(t) = h_1(t)$ and $x_2(t) = h_2(t)$ are state variables, $u(t) = [Q_{in,1}(t), Q_{in,2}(t)]^T$ is the input vector and $y(t) = h_2(t)$ is the output.

2. Given $a_1 = a_2 = 0.1m^2$, $A_1 = 20m^2$, $A_2 = 20m^2$, find the equilibrium state $(\bar{x}_1, \bar{x}_2)$ obtained with a constant input $\bar{u} = [1, 0]m^3/s$, approximating gravity acceleration as $g \approx 10m/s^2$.

3. Determine the linearized system (A, B, C, D) around the equilibrium state $(\bar{x}_1, \bar{x}_2)$.

4. In the programming language of your choice, simulate the two systems response to a input step of amplitude A of your choice and discuss your results.

QUESTION 1

Initially, we have to understand the bond between the fluid level in a tank and its flow. Given the volume of fluid in a tank as a function of time:

$$v(t) = h(t) \times \text{Area} \tag{1}$$

Taking the derivative of equation (1) with respect to time and a constant area:

$$\dot{v}(t) = \dot{h}(t) \times A \tag{2}$$

Given the equation (2), we can start modelling the system. Firstly, let's work with the flows of the tanks:

1. for the 1st tank:

$$\dot{v}_1(t) = -q_{out,1} + Q_{in,1} \tag{3}$$

2. for the 2nd tank:

$$\dot{v}_2(t) = q_{out,1} - q_{out,2} + Q_{in,2} \tag{4}$$

Using the equation (2) in the equations (3) and (4):

$$\begin{cases} \dot{h}_1(t) = \dfrac{1}{A_1}(-q_{out,1} + Q_{in,1}) \\ \dot{h}_2(t) = \dfrac{1}{A_2}(q_{out,1} - q_{out,2} + Q_{in,2}) \end{cases}$$

Since $q_{out,i}$ depends on the level of fluid in the tank$_i$:

$$q_{out,i}(t) = a_i\sqrt{2gh_i(t)} \tag{5}$$

From equation (5) we get the system of equations,:

$$\begin{cases} \dot{h}_1(t) = \dfrac{1}{A_1}(-a_1\sqrt{2gh_1(t)} + Q_{in,1}) \\ \dot{h}_2(t) = \dfrac{1}{A_2}(a_1\sqrt{2gh_1(t)} - a_2\sqrt{2gh_2(t)} + Q_{in,2}) \end{cases}$$

Using $x_1(t) = h_1(t)$ and $x_2(t) = h_2(t)$ as state variables, $u(t) = [Q_{in,1}(t), Q_{in,2}(t)]^T$ as the input vector, and $y(t) = h_2(t)$ as the output, we can rewrite the system of equations:

$$\begin{cases} \dot{x}_1(t) = \dfrac{1}{A_1}(-a_1\sqrt{2gx_1(t)} + Q_{in,1}) \\ \dot{x}_2(t) = \dfrac{1}{A_2}(a_1\sqrt{2gx_1(t)} - a_2\sqrt{2gx_2(t)} + Q_{in,2}) \end{cases} \tag{6}$$

QUESTION 2

In the equilibrium state (x_{1s}, x_{2s}):

$$\dot{x}_{1s} = \dot{x}_{2s} = 0$$

Using this property in the system of equations (6):

$$\begin{cases} 0 = \dfrac{1}{A_1}(-a_1\sqrt{2gx_{1s}} + Q_{in,1}) \\ 0 = \dfrac{1}{A_2}(a_1\sqrt{2gx_{1s}} - a_2\sqrt{2gx_{2s}} + Q_{in,2}) \end{cases}$$

From the previous system of equations we have (x_{1s}, x_{2s}):

$$\begin{cases} x_{1s} = \dfrac{Q_{in,1}^2}{2ga_1^2} \\ x_{2s} = \dfrac{(Q_{in,1} + Q_{in,2})^2}{2ga_2^2} \end{cases}$$

Given $a_1 = a_2 = 0.1m^2$, $A_1 = A_2 = 20m^2$, $\bar{u} = [1, 0]m^3/s$ and $g \approx 10m/s^2$, we find the equilibrium state:

$$(x_{1s}, x_{2s}) = (5, 5) \tag{7}$$

The code provided in Listing 2 produces the same result found manually. Using also the function *fsolve* in the code in Listing 4 (for Exercise 1.4) with $K = 1$ (the step input amplitude), we have the same result. It just need to change $Qin1$ (Listing 2) or K (Listing 4) to find new equilibrium points for differents inputs.

QUESTION 3

Since $f(x,u) = \dot{x}_{1,2}(t)$ are nonlinear functions, we have to linearize them to solve the linear system, and we can perform this by using the Taylor Series Expansion. This linear system must have similar characteristics and behavior to the non-linear system.
The Taylor series expansion of a function $f(x,u)$ around a equilibrium point (x_s, u_s), where $f(x_s, u_s) = 0$, is given by:

$$f(x,u) = f(x_s,u_s) + \left.\frac{\partial f}{\partial x}\right|_{x_s,u_s}(x - x_s) + \left.\frac{\partial f}{\partial u}\right|_{x_s,u_s}(u - u_s) + \text{higher order terms}$$

For the purposes of this model, we'll consider only the first 3 terms of the expansion (until the 1st order terms). So, for this case, we have the linearization of the function $f(x,u)$ around the choosen equilibrium point (x_s, u_s):

$$f(x,u) = \left.\frac{\partial f}{\partial x}\right|_{x_s,u_s}(x - x_s) + \left.\frac{\partial f}{\partial u}\right|_{x_s,u_s}(u - u_s)$$

And since the systems are connected, there's a bond between then that must be explicit in the linearization because of it's contribution to each other (u_i is the input of $u(t)$ in the tank i):

$$f(x,u) = \frac{\partial f}{\partial x_1}(x_1 - x_{1s}) + \frac{\partial f}{\partial u_1}(u_1 - u_{1s}) + \frac{\partial f}{\partial x_2}(x_2 - x_{2s}) + \frac{\partial f}{\partial u_2}(u_2 - u_{2s})$$

For $f(x,u) = \dot{x}_1(t)$:

$$\dot{x}_1(t) = \frac{-a_1\sqrt{2g}}{2A_1\sqrt{x_{1s}}}(x_1 - x_{1s}) + \frac{1}{A_1}(u_1 - u_{1s}) \tag{8}$$

For $f(x,u) = \dot{x}_2(t)$:

$$\dot{x}_2(t) = \frac{a_1\sqrt{2g}}{2A_2\sqrt{x_{1s}}}(x_1 - x_{1s}) + \frac{-a_2\sqrt{2g}}{2A_2\sqrt{x_{2s}}}(x_2 - x_{2s}) + \frac{1}{A_2}(u_2 - u_{2s}) \tag{9}$$

For simplification, let's write $x_i^* = x_i - x_{is}$ and $u_i^* = u_i - u_{is}$. With the parameters from question 2 we can rewrite the equations (8) and (9):

$$\begin{cases} \dot{x}_1^*(t) = -\dfrac{1}{200}x_1^* + \dfrac{1}{20}u_1^* \\ \dot{x}_2^*(t) = \dfrac{1}{200}x_1^* - \dfrac{1}{200}x_2^* + \dfrac{1}{20}u_2^* \end{cases}$$

Finally, we achieve the state-space model for the linearized system:

$$\begin{cases} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{200} & 0 \\ \frac{1}{200} & -\frac{1}{200} \end{bmatrix} \begin{bmatrix} x_1^*(t) \\ x_2^*(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{20} & 0 \\ 0 & \frac{1}{20} \end{bmatrix} \begin{bmatrix} u_1^*(t) \\ u_2^*(t) \end{bmatrix} \\ \quad y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \end{cases} \tag{10}$$

with the form:

$$\begin{cases} \dot{x} = \mathbf{A}x + \mathbf{B}u \\ y = \mathbf{C}x + \mathbf{D}u \end{cases} \tag{11}$$

where $\mathbf{D} = 0$.

QUESTION 4

After obtained the linearized system, we simulate both linear and non-linear responses. We used a step function with amplitude 1 for show the results in the following images.

For the linear system, we linearize the system around the equilibrium points, and since we are using step function with amplitude 1, these equilibrium points will be the same found in Equation 7. However, if the amplitude K is altered, it becomes necessary to recalculate the new equilibrium points to reflect the changed input. As demonstrated in Figures 3 and 5, selecting a higher amplitude results in a larger gain, causing the nonlinear response to take longer to reach the stationary point.

The Figure 2 shows how the fluid levels (x_1 and x_2) change over time in nonlinear system and in linear approximation.

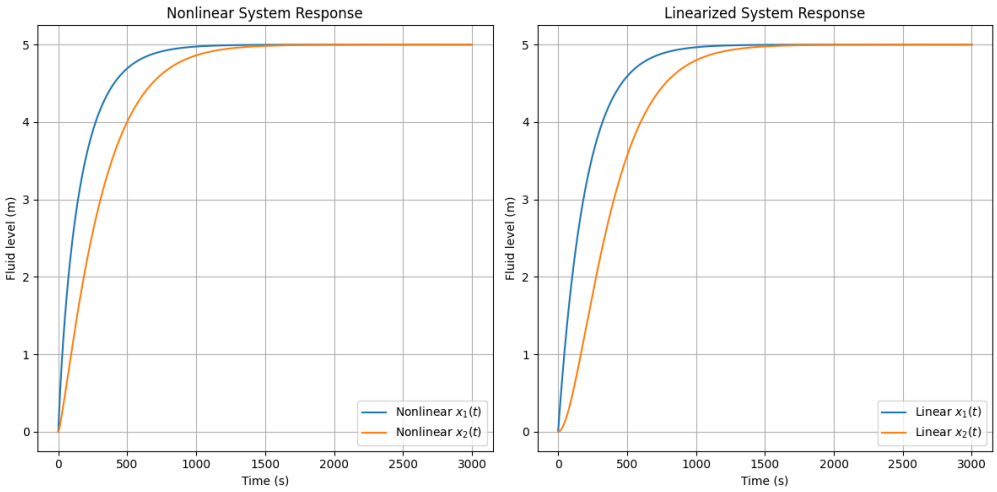

Figure 2: Nonlinear and system responses

Finally, the linearized system depicted in Figures 3 and 4 demonstrates that the linear responses for x_1 and x_2 reach the equilibrium point within a sufficient timeframe. The behavior of the linear system resembles that of a first-order response. At $K = 1$, the linear response closely matches the nonlinear system, evolving similarly over time.

However, as the amplitude K increases, this similarity decreases. This divergence is evident in Figure 5, which uses an amplitude of $K = 4$. Although both the linear and nonlinear responses ultimately converge to the same stationary point, the linear response reaches this point more quickly than its nonlinear counterpart.

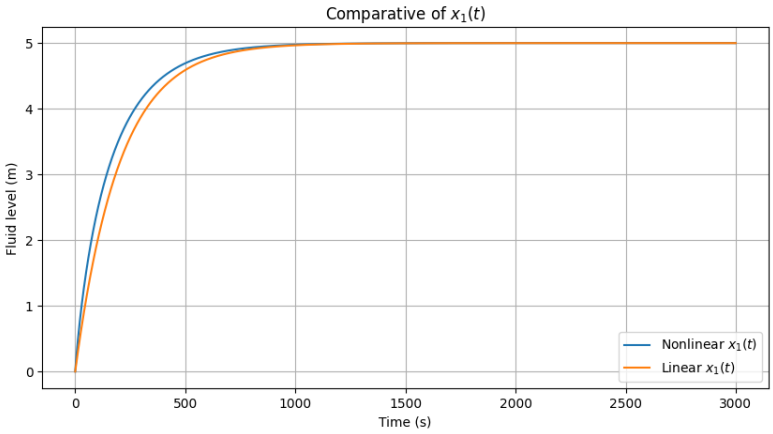

Figure 3: Comparison of state x_1

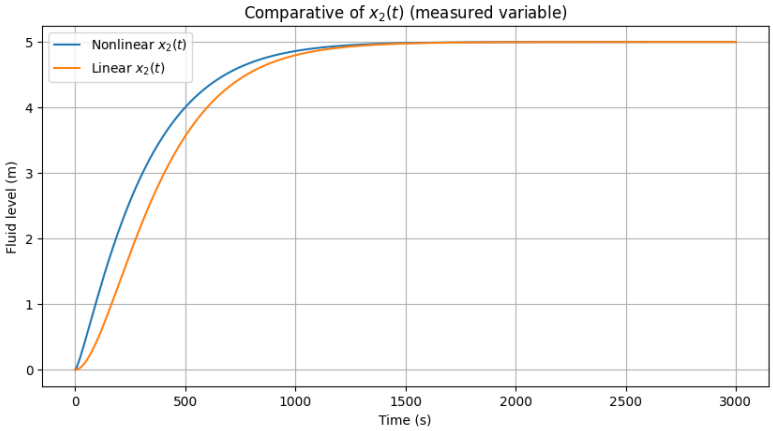

Figure 4: Comparison of the measured variable

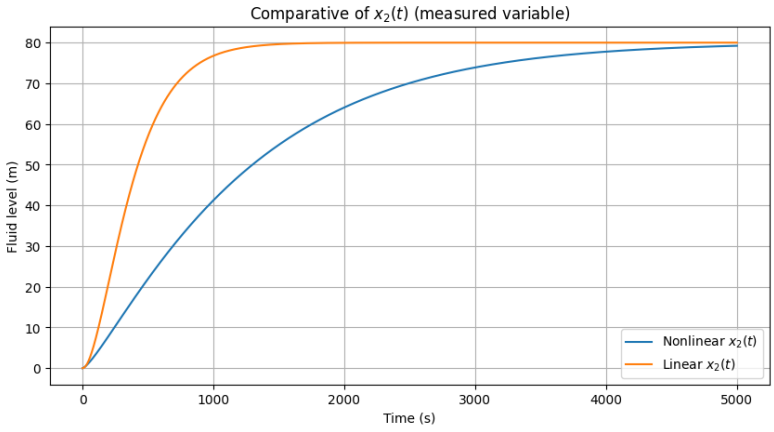

Figure 5: System response for amplitude K=4

TASK 2 - SYSTEM ANALYSIS

The objective of the second task of the project work is to analyse the dynamic two-tank system developed in the first task.

1. Considering the transfer function that describes the input-output system, plot the response of the system of a step input of amplitude A, defined based on your system.
2. By means of the time-response plot, find if possible:
   - the steady-state value
   - the % overshoot of the final value
   - the rise time
   - the settling time
3. Identify the poles and zeros of the linearized model (in the open-loop configuration). Plot the pole-zero map. Analyze the stability of the system by studying the map. Discuss the effect of poles and zeros on the process response and the important information you obtain from the map.
4. Define the possible manipulated (input) and controlled (output) variables and the corresponding properties of controllability and observability of the system.

QUESTION 1

With the state-space representation in (10), we can calculate $G(s)$ using the variables presented in (11) as:

$$\begin{aligned} G(s) &= \left[C\left(sI - A\right)^{-1}B + D\right] \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} s + 0.005 & 0 \\ 0.005 & -0.005 \end{bmatrix}^{-1} \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \\ &= \frac{\left(\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} s + 0.005 & 0 \\ 0.005 & s + 0.005 \end{bmatrix} \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} \right)}{(s + 0.005)^2} \\ &= \begin{bmatrix} \frac{25 \times 10^{-5}}{(s+0.005)^2} & \frac{0.05}{s+0.005} \end{bmatrix} \end{aligned}$$

For now and then, we will refer to G(s) as the transfer function that describes the input-output system in the vector $G(s)$:

$$G(s) = \frac{25 \times 10^{-5}}{s^2 + 0.01s + 25 \times 10^{-6}} \tag{12}$$

Figure 6 describes the step response with amplitude A = 1 of the system considering the transfer function that describes the input-output system defined based on our system:

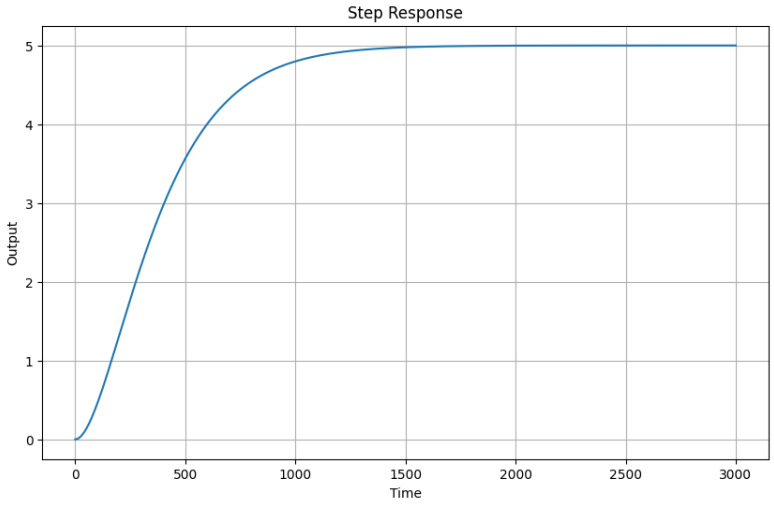

Figure 6: System response for amplitude A = 1

With the plot in figure 6 and the transfer function in (12), we can determine that the system represents a second order critically damped system.

QUESTION 2

In the time response plot, we can determine transient response specifications, such as steady-state value, rise time, and settling time. We can define and find the following specifications in the response:

- Steady-State value: The final value in which the system stabilizes.
- Percent Overshoot: The amount that the waveform overshoots the steady state, or final, value at the peak time, expressed as a percentage of the steady-state value.
- Rise Time (T_r): The time required for the system to transition from 10% to 90% of its final value.
- Settling Time (T_s): The duration needed for the system to reach and remain within 2% of its final value.

With these definitions, we can determine all the transient response specifications from the plot:

- Steady-State value: 5.0;
- Percent Overshoot: For a second order critically damped system, %OS = 0, since the response does not exceed the final value ($\zeta = 1$).;
- Rise Time (T_r): 671.467 seconds;
- Settling Time (T_s): 1166.817 seconds.

QUESTION 3

Figure 7 describes the pole-zeros map of the linearized model (open-loop configuration).

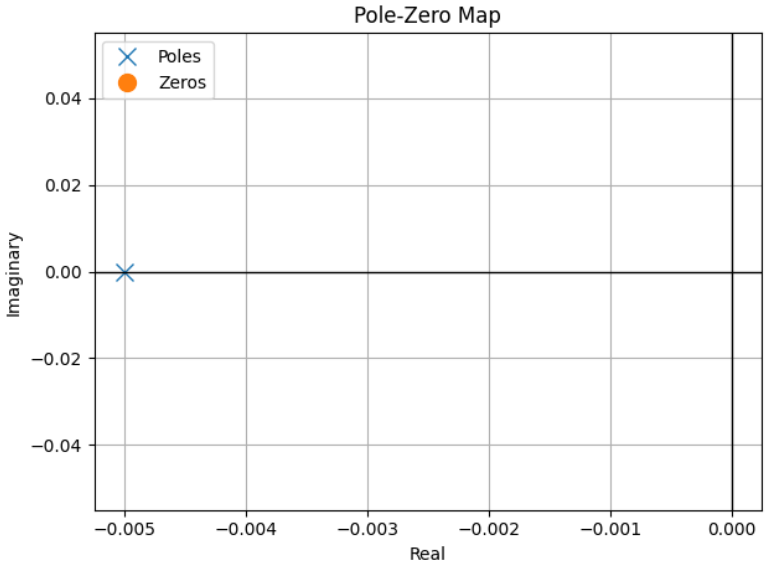

Figure 7: Pole-Zero Map

Analyzing the pole-zero map, we observe both the poles are located in the left half-plane of the $j\omega$-axis, indicating a stable, second order critically damped system with a slow response. In fact, we can assure that since the response it's reaching the steady-steady over time, as illustrated in figure 6. The absence of zeros and the presence of only two pole on the real axis simplify the analysis and control of this system.

This system exhibits characteristics typical of a first-order, Type 0 system:

- For constant inputs: The system will reach steady-state with no error, due to the pole's negative real-axis position.
- For step inputs: A finite, non-zero steady-state error will occur.
- For ramp inputs: An infinite steady-state error will result

The pole's location on left half-plane of the $j\omega$-axis guarantees stability and ensures the system can achieve steady-state for constant inputs. However, this configuration also means the system will struggle to accurately track changing inputs such as steps or ramps, resulting in steady-state errors for these input types.

This behavior underscores the system's nature as a Type 0 system, capable of steady-state accuracy for constant inputs but exhibiting limitations in tracking varying input signals without error.

QUESTION 4

The manipulated variables (input) of the system are the volumetric flow rates $Q_{in,1}$ and $Q_{in,2}$, and the controlled variable (output) is the fluid level h_2. Now, let's talk about the controllability and the observabillity of the system.

A system is considered controllable if we can find an input (or a sequence of inputs over time) that can move the system from any given initial state to any desired final state within a finite time. In order to determine controllability mathematically, we use a state-space representation of the system. For an nth-order plant, the state equation is typically expressed as

$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$

where x is the state vector, $\mathbf{A}$ is the system matrix, $\mathbf{B}$ is the input matrix, and u is the input vector. To determine the controllability of a system, we construct a matrix known as the controllability matrix, denoted as C_M:

$$C_M = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

A system is considered completely controllable if and only if the rank of C_M is equal to n, where n is the order of the system (number of states). This rank condition essentially means that the columns of C_M span the entire n-dimensional state space, implying that we can find a combination of inputs to reach any desired state.

For our system, we can determine the controllability matrix with the state-space representation in (10) with 2 states (n = 2):

$$C_M = \begin{bmatrix} 0.05 & 0 & -0.00025 & 0 \\ 0 & 0.05 & 0.00025 & -0.00025 \end{bmatrix}$$

With rank = 2. Thus, the system is controllable.

Simillarly, a system is considered observable if we can determine the complete internal state of the system from its outputs over a finite time interval. In other words, observability is about whether we can infer what is happening inside the system just by looking at its outputs. In order to determine observability mathematically, we use a state-space representation of the system, same as for controllability matrix:

$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$
$$y = \mathbf{C}x + \mathbf{D}u$$

Where $\mathbf{A}$ is the system matrix and $\mathbf{C}$ is the output matrix.
To determine the observability of a system, we construct a matrix known as the observability matrix, denoted as O_M:

$$O_M = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

The rank condition for observability is analogous to that for controllability. If the rank of O_M is equal to n (the order of the system), then the system is fully observable. This means that all states of the system can be inferred from the outputs over time.
For our system, we can determine the observability matrix with the state-space representation in (10) with 2 states (n = 2):

$$O_M = \begin{bmatrix} 0 & 1 \\ 0.005 & -0.005 \end{bmatrix}$$

With rank = 2. Thus, the system is observable.

TASK 3 - SYSTEM CONTROL

The objective of the third task of the project work is to analyse the dynamic two-tank system developed in the first task.

In this case, you must:

1. Define a simplified block diagram for the level control loop, by identifying the blocks for the level transmitter, the actuator and the system to be controlled.
2. Design a controller to obtain a step-response steady-state error of 10% without affecting the system's transient response appreciably.
3. Simulate the system with and without control using the programming language of your choice.
4. Comment your choices and the results.

QUESTION 1

We will define the following system:

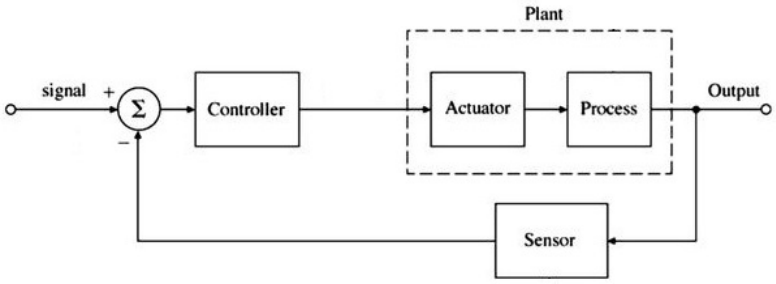

Figure 8: Simplified Block Diagram.

In which the controller will be a LAG compensator, the actuator will be the valve of the system and the process will be the transfer function defined in (12). We will apply a feedback with an ideal sensor - $H(s) = 1$ - for the controller adjust the response of the system, which will be the fluid level h_2.

QUESTION 2

Initually, we have to notice that, without the controller, the error in the response already is less than 10%:

$$e_o(\infty) = \lim_{s\to 0} \frac{1}{1+G(s)} = \frac{1}{1+K_{po}} = \frac{1}{1+10} = 0.090909... = 9.090909...\% \tag{13}$$

Thus, we will design a controller to obtain a step-response steady-state error of less than 1%. For this, lets improve the error in the order of 1800 times, which gives us:

$$e_n(\infty) \approx 5 \times 10^{-5} \tag{14}$$

With the error in (14), we can manage to find K_{pn}:

$$K_{pn} = \frac{1 - e_n(\infty)}{e_n(\infty)} \approx 20.000 \tag{15}$$

Since the improvement in K_p from the uncompensated system to the compensated system is the required ratio of the compensator zero to the compensator pole, or

$$\frac{z_c}{p_c} = \frac{K_{pn}}{K_{po}} \tag{16}$$

From the equation (13) and (15), we can find the ratio expressed in (16):

$$\frac{z_c}{p_c} = \frac{20.000}{10} = 2.000 \tag{17}$$

For the compensator, we have do add p_c next to the zero, for its angular effect being disconsidered. So, let $p_c = 10^{-6}$. With equation (17):

$$z_c = 2.000 \times 10^{-6} = 0.002$$

Finally, we have the following equation for the LAG controller:

$$LAG = \frac{K(s + 0.002)}{(s + 0.000001)} \tag{18}$$

With $K = 1$, for not affecting the system's transient response appreciably.

QUESTION 3

After simulating without and with the controller, we got the results in figure 9:

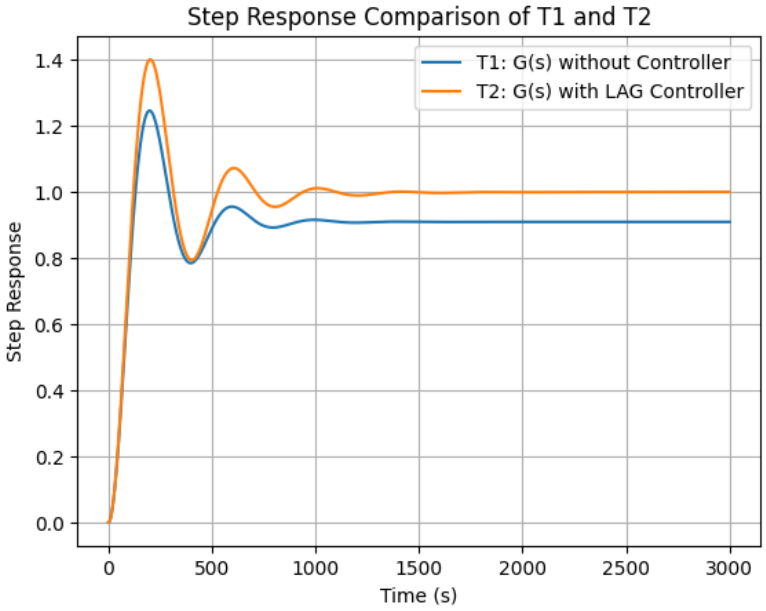

Figure 9: Step Response Without/With Controller.

QUESTION 4

LAG compensator was chosen because it is a control strategy that improves steady-state accuracy by reducing steady-state error, particularly for low-frequency inputs, without significantly affecting the system's transient response, as seen in figure 9. It achieved that by increasing the low-frequency gain while minimally impacting the higher-frequency behavior. It can be seen by visualising root-locus before (figure 10) and after (figure 11) the insertion of the LAG controller

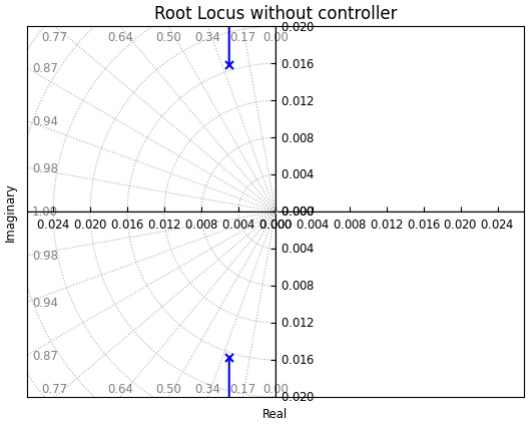

Figure 10: Root Locus of the System without LAG controller.

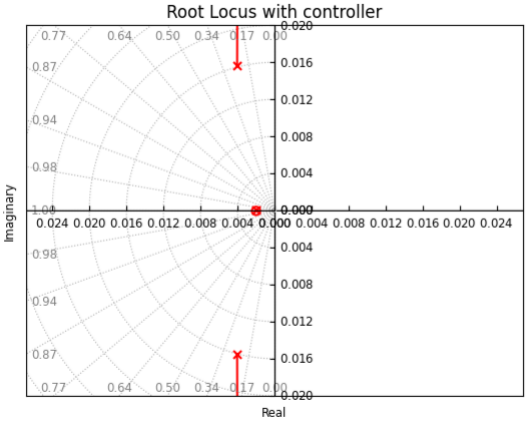

Figure 11: Root Locus of the System with LAG controller.

Additionally, the compensator enhanced system stability by adding damping, while kept control effort low. It was not so easy to design and implement but provided good results and it was worth the effort.

Appendix

TASK 1 - SYSTEM MODELLING

Question 1

Listing 1: Question 1

```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from sympy import symbols, Function, sqrt, Eq, diff, solve, Matrix, latex
from scipy import signal
from scipy import integrate
from scipy.optimize import fsolve
from IPython.display import display, Math

# Symbols for time, state variables, input variables, and parameters
t = symbols('t')
A1, A2, a1, a2, g = symbols('A_1 A_2 a_1 a_2 g')  # parameters

# State variables
x1_t = Function('x_1')(t)
x2_t = Function('x_2')(t)

# Input variables
Qin1_t = sp.Symbol('Q_{in,1}')
Qin2_t = sp.Symbol('Q_{in,2}')

# Torricelli's law for outflows
Qout1_t = a1 * sqrt(2 * g * x1_t)
Qout2_t = a2 * sqrt(2 * g * x2_t)

# State equations
dx1_dt = (Qin1_t - Qout1_t) / A1
dx2_dt = (Qin2_t + Qout1_t - Qout2_t) / A2

# Output equation
y_t = x2_t

display(Math(r'\dot{x}_1(t) = ' + latex(dx1_dt)))
print()
display(Math(r'\dot{x}_2(t) = ' + latex(dx2_dt)))
```

Question 2

Listing 2: Question 2

```
params = {a1: 0.1, a2: 0.1, A1: 20, A2: 20, g: 10, Qin1_t: 1, Qin2_t: 0}
eq1 = Eq(dx1_dt, 0)
eq2 = Eq(dx2_dt, 0)
eq1_subs = eq1.subs(params)
eq2_subs = eq2.subs(params)

# Solving the equations for state variables
equilibrium_solution = solve((eq1_subs, eq2_subs), (x1_t, x2_t))

display(Math(r'\bar{x}_1 = ' + latex(equilibrium_solution[0][0])))
print()
display(Math(r'\bar{x}_2 = ' + latex(equilibrium_solution[0][1])))
```

Question 3

Listing 3: Question 3

```
x2_eq, x2_eq = 5, 5 # equilibrium state
params.update({x1_t: x2_eq, x2_t: x2_eq})

# State vector and input vector
x = sp.Matrix([x1_t, x2_t])
u = sp.Matrix([Qin1_t, Qin2_t])

# System equations vector
f = Matrix([dx1_dt, dx2_dt])

# Calculate Jacobians
A = f.jacobian(x)
B = f.jacobian(u)
C = Matrix([[0, 1]])  # Output equation only relates to x2_t
D = Matrix([[0, 0]])  # No direct feed-through terms

# Substitute equilibrium values and parameters
A = A.subs(params)
B = B.subs(params)

display(Math('A = ' + latex(A)))
print()
display(Math('B = ' + latex(B)))
print()
display(Math('C = ' + latex(C)))
print()
display(Math('D = ' + latex(D)))
```

Question 4

Listing 4: Question 4.1

```
T = np.linspace(0, 3000, 2000)
K = 2 # amplitude
x0 = np.array([0., 0.]) # Initial conditions

# Nonlinear system dynamics
def system_dynamics(X, t):
  a1, a2, A1, A2, g, Qin1, Qin2 = 0.1, 0.1, 20, 20, 10, K, 0
  x1, x2 = X
  dx1dt = (Qin1 - a1 * np.sqrt(2 * g * x1)) / A1
  dx2dt = (Qin2 + a1 * np.sqrt(2 * g * x1) - a2 * np.sqrt(2 * g * x2)) /
      A2
  return [dx1dt, dx2dt]

# Solve the nonlinear system
nonlinear = integrate.odeint(system_dynamics, x0, T)
# Find the equilibrium points
x_eq = fsolve(system_dynamics, x0, args=(T,))

A_np = np.array([[-0.005,  0.], [0.005, -0.005]])
B_np = np.array([[0.05, 0.], [0., 0.05]])
C_np = np.array([[0., 1.]])
D_np = np.array([[0., 0.]])

# Define the state-space system
system = signal.StateSpace(A_np, B_np, C_np, D_np)

# Simulate linear system
_, y_linear, x_linear = signal.lsim(system, U=0, T=T, X0=x0-x_eq)
```

Listing 5: Question 4.2

```
# Plotting both results
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(T, nonlinear[:, 0], label='Nonlinear $x_1(t)$')
plt.plot(T, nonlinear[:, 1], label='Nonlinear $x_2(t)$')
plt.title('Nonlinear System Response')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(T, x_linear[:, 0] + x_eq[0], label='Linear $x_1(t)$')
plt.plot(T, x_linear[:, 1] + x_eq[1], label='Linear $x_2(t)$')
plt.title('Linearized System Response')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.legend()
plt.grid()
```

```
plt.tight_layout()
plt.show()
```

Listing 6: Question 4.3

```
# Comparing x1 (the output y)
plt.figure(figsize=(10, 5))
plt.plot(T, nonlinear[:, 1], label='Nonlinear $x_2(t)$')
plt.plot(T, x_linear[:, 1] + x_eq[1], label='Nonlinear $x_2(t)$')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.title('Comparative of $x_2(t)$ (measured variable)')
plt.legend()
plt.grid()
plt.show()
```

Listing 7: Question 4.4

```
# Comparing x1(t)
plt.figure(figsize=(10,5))
plt.plot(T, nonlinear[:, 0], label='Nonlinear $x_1(t)$')
plt.plot(T, x_linear[:, 0] + x_eq[0], label='Linear $x_1(t)$')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.title('Comparative of $x_1(t)$')
plt.legend()
plt.grid()
plt.show()
```

TASK 2 - SYSTEM ANALYSIS

Question 1

Listing 8: Question 1.1

```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from sympy import symbols, limit, solve, latex
from scipy import signal
from IPython.display import display, Math

!pip install control
import control as ctrl

# previous project work code / result
A = np.array([[-1/200,  0.], [1/200, -1/200]])
B = np.array([[1/20, 0.], [0., 1/20]])
C = np.array([[0., 1.]])
D = np.array([[0., 0.]])

x0 = np.array([0., 0.]) # Initial conditions
x_eq = np.array([5., 5.]) # equilibrium state

# state-space system
ss = signal.StateSpace(A, B, C, D)

# step response
t = np.linspace(0, 3000, 10000)
_, y_linear, x_linear = signal.lsim(ss, U=0, T=t, X0=x0-x_eq)

plt.figure(figsize=(10, 6))
plt.plot(t, y_linear + x_eq[1])
plt.title('Step␣Response')
plt.xlabel('Time')
plt.ylabel('Output')
plt.grid(True)
plt.show()

# Calculate the transfer function H(s) = C*(sI - A)^(-1)*B + D
s = sp.Symbol('s')
I = sp.eye(A.shape[0])  # Identity matrix
H_s = C * (s * I - A).inv() * B + D

# Simplify the transfer function
H_s = sp.simplify(H_s)

# Display the transfer function vector
H_s
```

Listing 9: Question 1.2

```
# Plotting both results
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(T, nonlinear[:, 0], label='Nonlinear $x_1(t)$')
plt.plot(T, nonlinear[:, 1], label='Nonlinear $x_2(t)$')
plt.title('Nonlinear System Response')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(T, x_linear[:, 0] + x_eq[0], label='Linear $x_1(t)$')
plt.plot(T, x_linear[:, 1] + x_eq[1], label='Linear $x_2(t)$')
plt.title('Linearized System Response')
plt.xlabel('Time (s)')
plt.ylabel('Fluid level (m)')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

Question 2

Listing 10: Question 2.1

```
steady_state = y_linear[-1] + x_eq[1]
print(f"steady_state value: {steady_state}")
```

Listing 11: Question 2.2

```
steady_state = y_linear[-1] + x_eq[1]

# Rise time calculation (10% and 90% criterion)
rise_start = 0.1 * steady_state
rise_end = 0.9 * steady_state
rise_start_index = np.argmax(y_linear + x_eq[1] >= rise_start)
rise_end_index = np.argmax(y_linear + x_eq[1] >= rise_end)
rise_time = t[rise_end_index] - t[rise_start_index]

print(f"Rise time: {rise_time:.3f} seconds")
```

Listing 12: Question 2.3

```
def find_settling_time_index(y, range_value, steady_state):
    for i in range(len(y) - 1, 0, -1):
        if not (steady_state - range_value <= y[i-1] <= steady_state +
            range_value):
            return i
    return 0

# Settling time calculation (2% criterion)
two_perc_range = 0.02 * steady_state
settling_time_index = find_settling_time_index(y_linear + x_eq[1],
    two_perc_range, steady_state)
settling_time = t[settling_time_index]
settling_value = y_linear[settling_time_index] + x_eq[1]

print(f"Settling time: {settling_time:.3f} seconds")
print(f"Value at settling time: {settling_value:.6f}")
print(f"Steady state value: {steady_state:.6f}")
```

Question 3

Listing 13: Question 3

```
# Separate numerator and denominator
num, den = H_s[1].as_numer_denom()

# Find zeros (roots of numerator)
zeros = sp.solve(num, s)

# Find poles (roots of denominator)
poles = sp.solve(den, s)

# Print results
print("Zeros:")
for zero in zeros:
    print(f"  {zero}")

print("\nPoles:")
for pole in poles:
    print(f"  {pole}")

# Stability check
stable = all(pole.is_complex and pole.as_real_imag()[0] < 0 for pole in
    poles)
print(f"\nSystem is {'stable' if stable else 'unstable'}")

# Pole-zero map
plt.plot(np.real(poles), np.imag(poles), 'x', markersize=10, label='Poles
    ')
plt.plot(np.real(zeros), np.imag(zeros), 'o', markersize=10, label='Zeros
    ')
plt.title(f'Pole-Zero Map')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.grid(True)
plt.legend()
plt.axhline(y=0, color='k', linewidth=1)
plt.axvline(x=0, color='k', linewidth=1)

plt.tight_layout()
plt.show()
```

Question 4

Listing 14: Question 4

```
controllability = ctrl.ctrb(A, B)
print("Controllability Matrix:")
print(controllability)

rank_controllability = np.linalg.matrix_rank(controllability)
print(f'\nRank of Controllability Matrix: {rank_controllability}')

observability = ctrl.obsv(A, C)
print("\nObservability Matrix:")
print(observability)

rank_observability = np.linalg.matrix_rank(observability)
print(f'\nRank of Observability Matrix: {rank_observability}')
```

TASK 3 - SYSTEM CONTROL

Question 1

Listing 15: Question 1

```
# Extract the numerator and denominator
num =           [2.5e-4]
den = [1, 0.01, 2.5e-5]

# Create the transfer function G(s) = 0.00025 / (s^2 + 0.01s + 2.5e-5)
G = ctrl.TransferFunction(num, den)

T1 = ctrl.feedback(G, 1)
t = np.linspace(0, 3000, 10000)
t, y1 = ctrl.step_response(T1, T=t)

err1 = (1 - y1[-1]) * 100
print(f"Steady-state error: {err1:.2f}%")
```

Question 2

Listing 16: Question 2

```
# Extract the numerator and denominator
num =           [2.5e-4]
den = [1, 0.01, 2.5e-5]

# Create the transfer function G(s) = 0.00025 / (s^2 + 0.01s + 2.5e-5)
G = ctrl.TransferFunction(num, den)

# Calculate the open-loop DC gain of G(s)
Kpo = ctrl.dcgain(G)

# Set the desired steady-state error (err = 0.00005)
err = 5e-5

# Calculate the proportional gain for the LAG controller (Kp)
Kpn = (1 - err) / err

# Define the pole of the LAG controller
pc = 1e-6

# Calculate the zero of the LAG controller (zc) based on the ratio Kpn/
    Kpo and pole pc
zc = (Kpn / Kpo) * pc

# Create the LAG controller transfer function: LAG(s) = (s + zc) / (s +
    pc)
LAG = ctrl.TransferFunction([1, zc], [1, pc])

# Create the closed-loop system T2 with feedback: T2 = LAG(s) * G(s) / (1
     + LAG(s) * G(s))
T2 = ctrl.feedback(LAG * G, 1)
t = np.linspace(0, 3000, 10000)
t, y2 = ctrl.step_response(T2, T=t)

err2 = (1 - y2[-1]) * 100
print(f"Steady-state error: {err2:.2f}%")
```

Question 3

Listing 17: Question 3

```
# Plotting both step responses on the same plot
plt.plot(t, y1, label='T1: G(s) without Controller')
plt.plot(t, y2, label='T2: G(s) with LAG Controller')

# Adding labels and title
plt.title('Step Response Comparison of T1 and T2')
plt.xlabel('Time (s)')
plt.ylabel('Step Response')
plt.grid(True)
plt.legend()

# Display the plot
plt.show()
```